On-line Submission and Testing of Programming Assignments

Mike Joy and Michael Luck,
Department of Computer Science,
University of Warwick,
COVENTRY,
CV4 7AL,
UK
email: {msj,mikeluck}@dcs.warwick.ac.uk

Abstract

The teaching of programming languages will often involve students being assessed by means of programming assignments. The administration of such assessments is a complex and demanding task which is compounded by increasing numbers of students.

At Warwick we have, in response, developed software which will allow such assignments to be submitted on-line. This paper discusses our experiences with the software and its implications for large group teaching.

1 Introduction

The numbers of students following computer programming courses — either within a computing degree or as part of another degree course — are increasing. At the same time, university staff are under considerable pressure to deliver such courses using available resources with maximum efficiency. A major (and critical) part of a programming course consists of the programming assignments, typically in the form of programs (or choices of programs) which the students are required to write. However, the process of testing and marking assessed pieces of software is very time-consuming and can be unreliable if attempted "by hand". Sharing the workload between several

members of staff is one way of coping with the extra workload generated by large student numbers, but leads to inconsistencies in marking.

Students put a large amount of effort into writing their programs, and expect — and deserve — thorough and accurate marking of their assignments, coupled with rapid turnaround so that they will receive useful feedback.

Fortunately, much of the testing and marking process has the potential to be automated. At Warwick we have been developing software which will allow students to submit programming assignments "on-line", and which will run those programs against test data. This paper reports our experiences after a year of using this software.

2 Necessity not Luxury

At Warwick, as at other universities, we have in the past relied on students handing in paper copies of programming assignments which are then marked by the appropriate lecturer. This method of assessing students' programming skills is flawed, and for several reasons:

- A decision must be reached for each submitted program as to whether or not it works. If that program is only available as "hard copy" it is *difficult* and *time-consuming* to decide whether or not it works. In addition, the final verdict may be *incorrect*.
- It is only possible to require students to demonstrate that a program has been tested on a relatively small set of test data, and it therefore cannot be tested on *unanticipated data*. Consequently, students will often tailor their programs to the test data rather than construct more general programs.
- If a student is required to submit test output from a program, that output can easily be *forged*.
- The *volume of paper* required is high, resulting in time spent physically handling it, together with possible errors when documents are accidentally mis-filed.

2.1 An Easy Solution?

A possible solution would be simply to require students to leave copies of their programs on a networked computer (or to hand in a disk containing a copy of the program). This is also problematic:

- Extreme care must be taken to ensure *privacy* of submitted programs.
- There are many ways in which the process can be *frustrated*, such as students failing to make their programs "readable" by the tutor.
- A program could contain a *"Trojan Horse"* which, when run by the tutor, might damage the security or integrity of the system.

- The lecturer is still required to spend *time* locating the program, compiling it and running it on suitable data before examining results.

2.2 Requirements of an On-line Submission System

In considering these problems, we identified a number of issues which would need to be addressed if a feasible system were to be introduced.

- The system must be *easy to use* — both for the students and for the lecturer setting and marking an assignment.
- *Security* is paramount — although we accept that complete security on a university computer network is probably unattainable, we needed to minimise any risks introduced by the system. These include students "hacking" into the system, students' programs accidentally (or deliberately) damaging the system, and the possibility of submitted documents becoming corrupted.
- The system must be sufficiently *flexible* to cope with different courses using different programming languages (both interpreted and compiled).

2.3 Packages Available from Elsewhere

There have been similar attempts to address some of these concerns, and we examined systems available from other institutions which might have assisted us. These included:

- *Ceilidh* — a system developed by Eric Foxley et al. at Nottingham [2]. Ceilidh contains many more features than our system, which we did not require, including on-line exercises and teaching aids.
- A package called *Submit* developed by Cameron Shelley at Waterloo.
- Other packages have been developed by, for instance, Collier at Northern Arizona University, Kay at UCLA, Isaacson and Scott at the University of Northern Colorado [5] and Reek at RIT [4].

All of these packages excited us. However, we finally took a decision that our own software was required. The principal reason for this was *security*. The other packages are written principally as collections of interrelated UNIX^{TM} shell scripts, a path we did not wish to follow. Rather, we wrote our software using C [1], conforming to the emerging UNIX standard known as POSIX [3]. Our software therefore maximises portability across different UNIX systems, and minimises being compromised by bugs in UNIX shell interpreters.

3 The Warwick Solution: *BOSS*

The package we developed, together with Chris Box, is named *BOSS* ("Bob's Online Submission System"), and contains a collection of programs which run under the UNIX operating system. It is designed specifically for courses which have a large number of students attending, and which are assessed by means of programming exercises. Assessed work must be in a form which can be specified very precisely (so that the output from students' programs can be compared with expected output); it is therefore not suitable for courses involving more generalised software design. Introductory programming courses in high-level computer languages are thus the typical target for *BOSS*.

The individual component programs of *BOSS* are as follows.

3.1 The program `submit`

This program reads a student's program, and stores it, encrypted, so that the lecturer can at a later date test it and mark it. It is a "user-friendly" program which will conduct a dialogue with the student to ensure that the correct submission is made. Preliminary checks will be carried out on the submitted program, to ensure that it appears to be in the correct language (for instance). The identity of the student submitting the program is verified.

An "acknowledgement of receipt" is sent to the student by email; this contains a code, generated by the "snefru" algorithm (the *Xerox Secure Hash Function*, ©Xerox Corporation 1989), which identifies the contents of their submission. A file only very slightly different (even by just one character) will generate a different code. Thus if a dispute arises, and it is claimed that a different file is assessed to that actually submitted, the code can be used to authenticate that file.

A student can also submit extra files (such as might contain documentation) which will also be available to the lecturer to mark.

3.2 The program `run_tests`

This program, which can only be run by a course tutor, will cause all submissions for a specified item of coursework to be run against a number of sets of data. Each student's programs will be copied to a newly-created directory, and run by a dummy usercode which has minimal system privileges, thus minimising the potential for the student's program to damage the system. Time and space limits are placed on the execution of a program, so as to prevent a looping program from continuing unchecked. The output from the student's program is checked against the expected output for each set of data, typically using a utility such as `diff`.

3.3 The program `mark`

This utility also can only be run by a course tutor. Initially the tutor is prompted to select one or more students. Each selected student's program is, in turn, made available to the tutor together with the output of `run_tests` on that program.

3.4 The program `testsubmit`

This program, which can be used by the students, will run the program which they are developing against *one* of the data sets on which it will eventually be tested, and under precisely the same conditions. Thus a student can check that their program will run correctly under the final testing environment. It is *not* a method for students exhaustively to test their program.

This program is important both for technical and for pedagogical reasons. Since the "Boss" system runs under UNIX, the UNIX environment is crucial to the correct running of a program, and many utilities require UNIX variables to be set correctly. In addition, programs may exist in several locations, and a given utility may have different versions. Many systems have, for example, two or more C compilers. So even if a student's program appears *to the student* to be working correctly, it is not always the case that it will work as expected when run by `run_tests`.

It is important to students as it provides a "confidence" hurdle which they can pass, by running their program on a (well-chosen) data set. They then have a reasonable expectation that their program is well on the way to completion.

3.5 Reliability

The system we have running is in the process of being tested, but so far we have encountered no problems in the *BOSS* software, although some "features" of the version of the operating system it is running on have caused complications. We have run the system so far on 3 courses, two involving Pascal and one which covered UNIX Shell programming, and each attracting roughly 150 students. No student has yet broken the security of the system.

3.6 Options

The system can be tailored so that its behaviour can be changed. For example, the default system limits on a student's program during testing can be changed. The number of times a student is allowed to submit a program can be restricted. The availability of the `submit` command can be restricted to a specified list of students.

4 Marking versus Testing

The *BOSS* system is a tool to allow students to *submit* assignments, and for those programs to be *tested* automatically. It is *not* an automated marking system. It is

the responsibility of the individual lecturer to provide a marking scheme which takes account of the results produced by *BOSS*, together with all other factors which may be regarded as important (such as program style, commenting, etc.).

Action that should be taken when a student's program does *not* pass one or more of the tests on which it is run, is again the lecturer's responsibility. It may be desirable to award marks for a partially working program — however *BOSS* does not address that problem.

5 Benefits

The *BOSS* system has provided us with a number of benefits, including those following.

- *Large numbers* of students can be handled efficiently by the system.
- *Security* of assignment submission is assured — programs submitted cannot be copied by other students, and the possibility of paper submissions being accidentally "lost" is removed.
- Secretarial staff do not need to be employed at deadlines to collect assignments, thus *more efficient use is made of secretarial time*.
- The *time needed to mark* an assignment is reduced considerably.
- The *volume of paperwork* involved is reduced to (almost) zero both for the lecturer and for administrative and secretarial staff.
- The *accuracy* of marking and testing is improved, and consequently the confidence enjoyed by the students in the marking process.
- *Consistency* of marking is improved, especially if more than one person is involved in the marking process.
- There is potential for *further checks* to be built into the system, such as automatic checks for plagiarism.

There have been few teething troubles, and those there have been relate to "features" of and bugs in the operating system under which the system is running.

6 Program Specification

In order for a system such as *BOSS* to indicate correctly whether or not a student's program works, assignments must be specified very accurately. This is because utilities such as `diff`, which can be used to compare a program's output to that which is expected, are precise formal tools. This places a burden on the lecturer to ensure that the assignments are very carefully worded. Experience has highlighted the difficulty of doing this.

It follows from the necessity of exact specification for a programming task that there is less scope available to students for novel solutions. It might be argued that this is a potential demerit of the system, but conversely it encourages students to adopt the rigour and precision expected of a modern engineering discipline.

Another drawback of this approach is that by specifying a program simply in terms of expected output for a given input, the internal structure of a program cannot be checked automatically. An example where this would be important might be a program to perform bubblesort on a list of numbers. The algorithm employed by a student must be checked by the lecturer when marking, by examining the program source code.

7 Student Response

We sought the views of our Computer Science students on *BOSS*, by means of a questionnaire. These were generally favourable, and most students considered it an easy system to use. The ability to use `testsubmit` to check the conformance of their programs to the specification was also widely appreciated.

The principal concerns expressed fell into two categories. First of all, the user interface is somewhat raw — and in particular when a program *fails* `testsubmit` the messages delivered are not very lucid. This is a fairly simple task to correct.

The second — and more interesting — criticism is that the output expected was *too precisely specified*. *BOSS* is far too "fussy". All the students who have used the system have been first year undergraduates, many of whom have had considerable programming experience prior to joining our course. Many of them are thus used to programming in an unstructured fashion. We wonder to what extent these concerns are fuelled by a "culture shock", simply not being used to being required to follow precise specifications.

8 Future Developments

As it stands, the systems is functioning well. The generally favourable student response has already been discussed above, and this is expected to improve once the culture of automatic submission has been established within the Department. In addition, lecturers and tutors have also found the system to be simple and easy to use, and marking times have been reduced significantly with a corresponding increase in consistency throughout.

We hope to extend the system to include extra facilities. We are currently designing an extension to "submit" which will perform automatic checks to indicate possible instances of plagiarism.

The user interface is at present quite rudimentary. In the next few months we intend to produce windowed software for the `mark` program, which will help to speed up marking of assignments even further.

Though these extensions are not yet complete, results so far have been highly encouraging, and the significant beneficial effects of using the system have already been felt by students, academic staff and secretarial staff alike.

9 Further Information

Technical specifications may be obtained by mailing `boss@csv.warwick.ac.uk`, together with details on how institutions may obtain a copy of the software.

References

[1] AMERICAN NATIONAL STANDARDS INSTITUTE (1990) Programming Language — C, New York. Standard ANSI X3.159-1989.

[2] BENFORD, S D, BURKE, K E, FOXLEY, E, MOHD ZIN, A & GUTTERIDGE, N H (1993), The Design of Ceilidh Version 2, Technical Report, Learning Technology Research, Computer Science Department, University of Nottingham.

[3] IEEE
(1990) Information Technology – Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language], New York. Standard IEEE 1003.1-1990.

[4] REEK,
K A (1989) The TRY System — or — How to Avoid Testing Student Programs. ACM SIGCSE Bulletin, 21(1) 112-116.

[5] ISAACSON, P C & SCOTT, T A
(1989) Automating the Execution of Student Programs, ACM SIGCSE Bulletin, 21(2) 15-22.